CS 1110 Spring 2020, Assignment 1: Write your own CourseGrab (Get open/closed/waitlist status from the registrar's live webpages)

Updates:

- Tuesday Feb 11, 9:30am: A1 file `a1_first.py`, typo in `test_after()` test case #10: it was missing the second argument in the call to `assert_equals`
- Mon Feb 10, 7pm: third precondition added to `first_between` in `a1_second.py`; see page 6.
- Sun Feb 9, 6:29pm: A1 file `a1_first.py`, had a typo in `test_after()` test case #5: it was missing the line `testcase.assert_equals(' A ', result)` at the end.

The online files have been updated, but if you had already started working on them, do *not* re-download them: don't risk over-writing the work you already did!

CS 1110 Spring 2020, Assignment 1: Write your own CourseGrab (Get open/closed/waitlist status from the registrar's live webpages)*†

http://www.cs.cornell.edu/courses/cs1110/2020sp/assignments/assignment1/a1.pdf

Navigating links in this pdf. Text in any shade of blue in this document is a click-able link.

Watch the course website for any announcements or updates.

Contents

1 Rules **1**
- 1.1 How To Register a Partner (You Only Get One) . . . 1
- 1.2 What Collaborations Are (Dis-)Allowed And How To Document Them . . . 2
- 1.3 Python You Are NOT Allowed To Use In This Assignment . . . 2

2 Deadlines **2**
- 2.1 Is there going to be a chance to revise in response to grader feedback? . . . 2

3 Task Overview: Extracting Roster Information **3**
- 3.1 What we provide . . . 5
  - 3.1.1 Files description . . . 5
  - 3.1.2 Call structure . . . 5
- 3.2 Your task: the short version . . . 5

4 Iterative Development (How to Work Through the Assignment) **5**

5 Grand Finale **8**

6 Code Cleanup **8**

7 Pre-Submission Checklist and What to Submit **8**

1 Rules

1.1 How To Register a Partner (You Only Get One)

You may work alone or with exactly one other person.

If you are partnering, **the two of you must form a group on the CS1110 course management system CMS BEFORE submitting, which will link your submission "portals"**. More details are in Section 2.

If your partnership dissolves, follow the instructions in our Academic Integrity policies' item on the "group divorce scenario".

*Authors: Lillian Lee, with some instructions derived from previous assignments by Walker White and David Gries and formatting initially created by Stephen McDowell.

†Assignments will be motivated by real-life use-cases, such as students wanting to know whether a lecture or section opens up. We admit that we had wanted A1 to involve Iowa Caucus results as a current-events application of your newly acquired Python skills, but alas, that didn't work out, slightly delaying release of this assignment.

1.2 What Collaborations Are (Dis-)Allowed And How To Document Them

The full policy is on the course Academic Integrity page, but we re-state here **the main principles**, where "you" means you and, if there is one, your one CMS-registered group partner,

1. Never look at, access or possess any portion of another group's code in any form. (This includes code written on a whiteboard.)
2. Never show or share any portion of your code in any form to anyone except a member of the course staff.
3. Never request solutions from outside sources, for example, on online services like StackOverflow.
4. DO specifically acknowledge by name all help you received, whether or not it was "legal" according to (1)-(3).

Rule (4) means "cite your sources": the header comments of your code must accurately describe the entire set of people and sources[1] that contributed to the code that is submitted.

Example:[2]

```
# a1_second.py
# KDF4 and LJL2
# Sources/people consulted: discussed strategy for after() with Walker White
# Feb 12, 2020
```

1.3 Python You Are NOT Allowed To Use In This Assignment

You may not use, and do not need, any Python constructs not yet covered by the labs, lectures, or posted lecture slides by the time this assignment was released.[3] We want you to demonstrate your skills with the Python we have taught so far.

2 Deadlines

There are several deadlines on Wednesday, February 19.

- If you are partnering: well before submission, follow the instructions in the "How to form a group" instructions on the course CMS usage guide, linked to on our Resources page. Both parties need to act on CMS (http://cmsx.cs.cornell.edu) in order for the grouping to take effect.
- By 2pm on Wednesday, February 19, submit whatever you have done at that point to CMS, following steps 1-3 in the "Updating, verifying, and documenting assignment submission" section of our CMS usage guide. It is OK if you haven't finished yet; CMS lets you update submissions until the final deadline.
- By 11:59pm on Wednesday, February 19, make your final files submission on CMS, once again following the aforementioned "updating/verifying/documenting" steps 1-3.

The 2pm checkpoint on Wednesday, February 19 provides you a chance to alert us during business hours if any problems arise. Since you've been warned to submit early, do not expect that we will accept work that doesn't make it onto CMS on time, for whatever reason, including server delays stemming from many other students trying to submit at the same time as you.[4]

Of course, if some special circumstances arise, contact the instructor(s) immediately.

2.1 Is there going to be a chance to revise in response to grader feedback?

Stay focused on hitting the deadlines listed above. Later, we'll talk again.

[1] Other than the course staff or course materials.

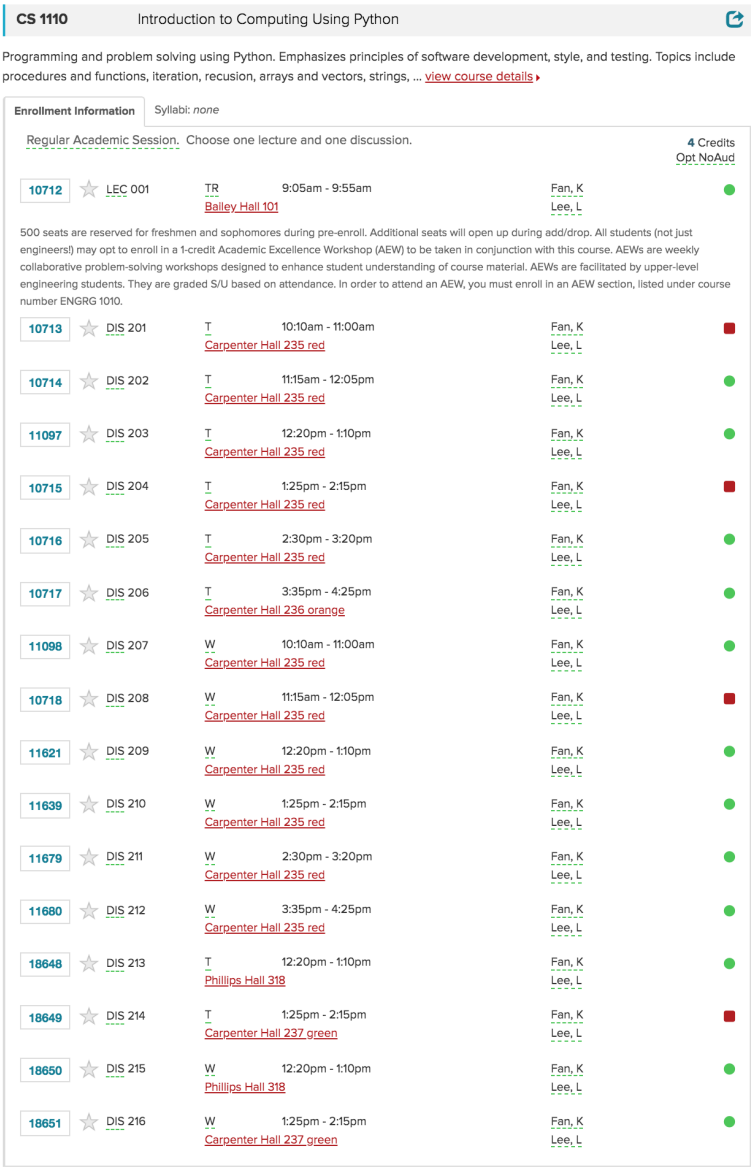
[2] We aren't asking for your names, on the principle that grading an assignment shouldn't require knowing the identity of the author(s). But we do want to have NetID identifiers so we don't get confused when grading many files at once.

[3] So, no `if`s, no loops, etc., even if for some reason you know what those are.

[4] There are no so-called "slipdays" and there is no "you get to submit late at the price of a late penalty" policy.

3 Task Overview: Extracting Roster Information

When you look at a Cornell course roster page such as https://classes.cornell.edu/browse/roster/SP20/subject/CS:



CS 1110 Introduction to Computing Using Python

Programming and problem solving using Python. Emphasizes principles of software development, style, and testing. Topics include procedures and functions, iteration, recusion, arrays and vectors, strings, ... view course details

Enrollment Information | Syllabi: *none*

Regular Academic Session. Choose one lecture and one discussion. **4** Credits, Opt NoAud

10712	LEC 001	TR	9:05am - 9:55am	Fan, K Lee, L	
		Bailey Hall 101			

500 seats are reserved for freshmen and sophomores during pre-enroll. Additional seats will open up during add/drop. All students (not just engineers!) may opt to enroll in a 1-credit Academic Excellence Workshop (AEW) to be taken in conjunction with this course. AEWs are weekly collaborative problem-solving workshops designed to enhance student understanding of course material. AEWs are facilitated by upper-level engineering students. They are graded S/U based on attendance. In order to attend an AEW, you must enroll in an AEW section, listed under course number ENGRG 1010.

10713	DIS 201	T	10:10am - 11:00am	Fan, K Lee, L	
		Carpenter Hall 235 red			
10714	DIS 202	T	11:15am - 12:05pm	Fan, K Lee, L	
		Carpenter Hall 235 red			
11097	DIS 203	T	12:20pm - 1:10pm	Fan, K Lee, L	
		Carpenter Hall 235 red			
10715	DIS 204	T	1:25pm - 2:15pm	Fan, K Lee, L	
		Carpenter Hall 235 red			
10716	DIS 205	T	2:30pm - 3:20pm	Fan, K Lee, L	
		Carpenter Hall 235 red			
10717	DIS 206	T	3:35pm - 4:25pm	Fan, K Lee, L	
		Carpenter Hall 236 orange			
11098	DIS 207	W	10:10am - 11:00am	Fan, K Lee, L	
		Carpenter Hall 235 red			
10718	DIS 208	W	11:15am - 12:05pm	Fan, K Lee, L	
		Carpenter Hall 235 red			
11621	DIS 209	W	12:20pm - 1:10pm	Fan, K Lee, L	
		Carpenter Hall 235 red			
11639	DIS 210	W	1:25pm - 2:15pm	Fan, K Lee, L	
		Carpenter Hall 235 red			
11679	DIS 211	W	2:30pm - 3:20pm	Fan, K Lee, L	
		Carpenter Hall 235 red			
11680	DIS 212	W	3:35pm - 4:25pm	Fan, K Lee, L	
		Carpenter Hall 235 red			
18648	DIS 213	T	12:20pm - 1:10pm	Fan, K Lee, L	
		Phillips Hall 318			
18649	DIS 214	T	1:25pm - 2:15pm	Fan, K Lee, L	
		Carpenter Hall 237 green			
18650	DIS 215	W	12:20pm - 1:10pm	Fan, K Lee, L	
		Phillips Hall 318			
18651	DIS 216	W	1:25pm - 2:15pm	Fan, K Lee, L	
		Carpenter Hall 237 green			

you can see which course's lectures and sections are open, closed, waitlisted, and so on, but it would be nice to have a more efficient way to look up such information and a more compact presentation.

In this assignment, you'll write functions called by a function we created that pulls such information from the online course roster. When done, you'll be able to run the program `get_components.py`[5] on live Cornell Spring 2020 roster pages to have interactions like what's shown on the next page.[6]

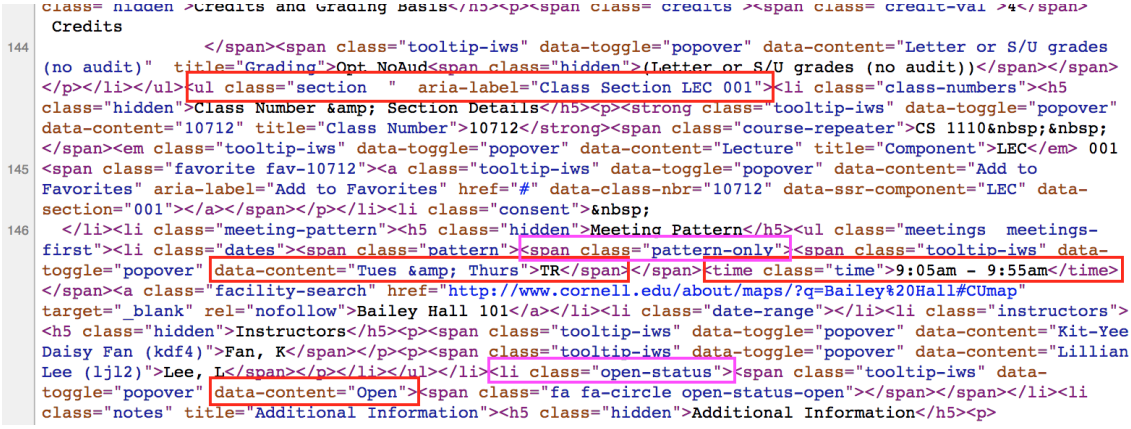
[5] Lectures, sections, and labs are "components" in registrar-speak.

[6] So, when you complete this assignment, you might want to check whether the final output you get matches what's shown in this document.

```
(base) llee@ushuaia: ~/cs1110/2020sp/a1 > python get_components.py
Enter a subject rubric, such as "CS" or "AEM", to look up in the Spring 2020 roster
(just hit return for "CS"): CS
The webpage for CS Spring 2020 has been loaded.

Note that this program does NOT refresh its data;
if the webpage gets updated, you'll need to restart this program to get the new info.

Enter the class number for subject CS, such as 1110, or "q" to quit: 1110

Here are all components whose times I found (there may not be any).
LEC 001 TR 9:05am - 9:55am Open
DIS 201 T 10:10am - 11:00am Closed
DIS 202 T 11:15am - 12:05pm Open
DIS 203 T 12:20pm - 1:10pm Open
DIS 204 T 1:25pm - 2:15pm Closed
DIS 205 T 2:30pm - 3:20pm Open
DIS 206 T 3:35pm - 4:25pm Open
DIS 207 W 10:10am - 11:00am Open
DIS 208 W 11:15am - 12:05pm Closed
DIS 209 W 12:20pm - 1:10pm Open
DIS 210 W 1:25pm - 2:15pm Open
DIS 211 W 2:30pm - 3:20pm Open
DIS 212 W 3:35pm - 4:25pm Open
DIS 213 T 12:20pm - 1:10pm Open
DIS 214 T 1:25pm - 2:15pm Closed
DIS 215 W 12:20pm - 1:10pm Open
DIS 216 W 1:25pm - 2:15pm Open
......

Enter another course number,  or "q" to quit:  1112

Here are all components whose times I found (there may not be any).
LEC 001 TR 11:15am - 12:05pm Open
DIS 201 T 12:20pm - 1:10pm Open
DIS 202 T 1:25pm - 2:15pm Open
DIS 203 T 2:30pm - 3:30pm Open
DIS 205 W 10:10am - 11:00am Open
DIS 206 W 11:15am - 12:05pm Open
DIS 207 W 12:20pm - 1:10pm Open
DIS 208 W 1:25pm - 2:15pm Open
DIS 209 W 2:30pm - 3:20pm Open
......

Enter another course number,  or "q" to quit: q
(base) llee@ushuaia: ~/cs1110/2020sp/a1 >
```

The key to this process is that many webpages are really just big collections of special strings that your browser displays using formatting information in those strings. You can view the underlying string for a given webpage by using the "view source" functionality of your browser.[7] Here is an excerpt of the source (underlying string) for the webpage above:

[7] Chrome: View » Developer » View Source. Firefox: Tools » Web Developer » Page Source. Safari: Develop » Show Page Source. Or, right-click or ctrl-click in the browser window often brings up a menu with an option to view page source.



We've highlighted with red boxes the locations of the information needed by `get_components.py`. The "label" for the main lecture is in the `class="section "` box. The meeting days are in the `data-content` box following the magenta `class="pattern-only"` box.[8] And so on.

3.1 What we provide

3.1.1 Files description

Create a new directory on your computer. Download the following into that new directory.

http://www.cs.cornell.edu/courses/cs1110/2020sp/assignments/assignment1/get_components.py
http://www.cs.cornell.edu/courses/cs1110/2020sp/assignments/assignment1/a1_first.py
http://www.cs.cornell.edu/courses/cs1110/2020sp/assignments/assignment1/a1_second.py
http://www.cs.cornell.edu/courses/cs1110/2020sp/assignments/assignment1/testcase.py

We've written the entire program `get_components.py` for you! But it won't work as expected yet, because it makes calls to an unfinished function in file `a1_second.py`.

`a1_second.py` is a "skeleton" containing the function-definition headers and docstring specifications you'll need — **don't change those** — but with the function bodies consisting only of the do-nothing command `pass`.

We've also given you a partially-completed file `a1_first.py` for testing the functions in `a1_second.py`. **It has some purposely erroneous test cases in it**; more on that later.

3.1.2 Call structure

- The program in file `get_components.py` repeatedly calls function `report_section()` in file `a1_second.py`.
- Function `report_section()` in `a1_second.py` should call "helper" function `first_between()`[9] which itself should call `after()`.
- The testing functions in `a1_first.py` should each call the corresponding functions from `a1_second.py` multiple times.

3.2 Your task: the short version

In a nutshell, you must **fix** and **complete** files `a1_first.py` and `a1_second.py`, following all directions given in comments prefaced with the all-caps word "`STUDENTS`" as well as in this document.

You are allowed to write your own helper functions, but if you do, you must (a) provide clear specification docstrings for them, and (b) provide adequate testing for them in `a1_first.py`

4 Iterative Development (How to Work Through the Assignment)

As outlined in Section 3.1.2, there are dependencies between the functions you will write. The wisest course of action is to work on the basic functions first, and make sure they are correct, before moving on to the functions that build on that basis.

[8] We need the magenta info because there are a number of occurrences of `data-context` elsewhere in the excerpt shown.
[9] In fact, we are expecting it to do so multiple times.

Hence, develop and test the functions in `a1_second.py` one at a time, in order. For each function, do the following.

1. **Carefully read the specification for the function.**[10] [11]

```
def after(text, tag):
    """Returns: the part of `text` just after at the 1st occurrence of `tag`.

    Preconditions:
        `text` [str]: contains at least one instance of `tag`
        `tag`  [str]: length > 0

    Examples:
        after('start <a id="c111"> this that', '<a id="c111">')
            --> ' this that'
            (Note the single space at the beginning.)

        after('start <a id="c111"> this that the other', 'other')
            --> ''
    """
```

Mon, Feb 10, 7pm: third precondition added

```
def first_between(text, start_str, end_str):
    """Returns substring of `text` occurring between the 1st occurrence of
    `start_str` and the first following occurrence of `end_str`.
    Preconditions:
        `text` [str]: length > 0.
        `start_str` and `end_str` [str]: both non-empty and occur in `text`.
         At least one `end_str` appears after a `start_str` in `text`.


    Examples:
        first_between('A1A2A3bA4ccccA5', 'A', 'b') ---> '1A2A3'
        first_between('hi)bye(x)(a+b)', '(', ')')  --> 'x'
            (Note that there was an end_str before start_str.)

        first_between('<a href="x">yes</a><span>toodles</span>', '<span>', '<')
            --> 'toodles'
    """
```

[10] This step makes sure you know what a function is supposed to do. You don't get points for writing correct code for the wrong task!

[11] Backquotes are used to visually distinguish variable names, like this: `` `tag` ``. We're not using angle brackets (as in lecture) because we'll be working with html strings, which themselves contain angle brackets.

```
def report_section(s):
    """Returns string '<component> <meeting day(s)> <time> <status>'.

    Preconditions:
    `s` corresponds to a single class component (=lecture/section)
    that isn't TBA, having a pattern like this
        '<ul class="section ... aria-label="Class Section DIS 202"
         ... class="pattern-only"><span ...>T</span>
         ... class="time">11:15am - 12:05pm</time>
         ... <li class="open-status"> ... data-content="Open"...'
    where the following substrings occur exactly once:
        '<ul class="section'
        'class="pattern-only"><span'
        'class="time">'
        <li class="open-status">

    For this example, this function returns 'DIS 202 T 11:15am - 12:05pm Open'.
    """
```

2. **Look at the test cases in `a1_first.py` and fix the bad ones.**[12] [13]

   Function `test_after()` contains *at least* two bad test cases.[14]

   - For a test case where the "expected answer" is wrong, add the comment `# ... STUDENT-FIXED ERROR ...` under the comment that numbers the test case; comment out the incorrect `assert_equals` call; and add the fixed call below, like this:

     ```
     # 0. tag at beginning
     # ... STUDENT-FIXED ERROR ...           <--- added
     result = a1_second.after('sta', 'st')
     # testcase.assert_equals('', result)    <--- commented out
     testcase.assert_equals('a', result)     <--- fix added
     ```

   - For a test case where the situation should not have been tested, add the comment `# ... STUDENT-DELETED CASE ...` under the comment that numbers the test case; add a comment giving your reasoning, and comment out the entire test case, like this:

     ```
     # 1110. tag is an int
     # ... STUDENT-DELETED CASE ...           <--- added
     # ... REASON: violates precondition      <--- added
     # result = a1_second.after('123', 2)     <--- commented out
     # testcase.assert_equals('3', result)    <--- commented out
     ```

3. **Add missing representative test cases for that function in the appropriate place in `test_after()` and `test_first_between()`.**[15] You want cases that represent valid inputs, but exhibit different aspects of the problem the function is trying to solve, so that using your suite of test cases can catch different types of bugs.

   For each test case you add, include in a comment a short justification of what the test case represents.

[12] We want to make sure you examine and understand test cases — and thus, the specifications of the functions they are testing — even though we just introduced unit tests in Lecture 6. To this end, we are not only supplying you with a number of test cases, so you have many examples to look at; but purposely introducing some erroneous ones to force you to examine them carefully.

[13] We guarantee that there are no (intentional) errors in the "expected answers" in `test_first_between` or `test_report_section`.

[14] Just to be clear: maybe there are just two, maybe there are three, maybe there are more...

[15] We have constructed enough test cases for you for `test_report_section()`, so you don't need to add any more. You're welcome.

4. **Write the function body in `a1_second.py`**[16] [17] [18]

5. **Run python on the script `a1_first.py`.**[19] If errors are revealed with the function you're currently working on, fix them and re-test.

5 Grand Finale

If you're convinced that your code is correct, you should be able to run Python on the file `get_components.py`[20] and reproduce the interaction in Section 3.

6 Code Cleanup

Before submitting, ensure your code obeys the following.[21]

- Lines are short enough (~80 characters) that horizontal scrolling is not necessary.
- You have indented with spaces, not tabs
- Functions are separated from each other by at least two blank lines.
- You have removed any debugging `print` statements.
- You have removed all `pass` statements.
- You have removed "instruction" comments, such as "`# IMPLEMENT THIS FUNCTION`".
- If you added any helper functions, these have good docstring specifications and you have put sufficient testing code for your functions in `a1_second.py` .

7 Pre-Submission Checklist and What to Submit

The files to submit to CMS (http://cmsx.cs.cornell.edu) are `a1_first.py` and `a1_second.py`.[22]

Make sure the following are all true before you submit.

1. You've changed the header comments in all files to list the entire set of people and sources that contributed to the code.
2. You (and your partner) have included your NetIDs in the header of all files.
3. The date in the header comments has been changed to when the files were last edited.
4. You have set your CMS notifications settings to receive email regarding grade changes, and regarding group invitations.
5. (reminder) If working with a partner, you have grouped on CMS. (One has invited on CMS, and one has accepted on CMS.)

[16] Hint: If the specification says to return something, you need a `return` statement returning something of the correct type.

[17] Another hint: double-check that if the instructions said to call a certain helper, that you did indeed use that helper function.

[18] Also: the functions in `get_components.py` do some string processing, so you may find it useful to look in that file for inspiration/examples. But it is definitely not necessary to do so, and if you do choose to check that file out, don't be intimidated by the Python in there that you don't know (yet).

[19] At the command prompt, *not* the `>>>` Python interactive prompt, enter `python a1_first.py` .

[20] At the command-shell prompt, enter `python get_components.py` .

[21] These requirements up speed up the process of reading/grading hundreds of files.

[22] Do not submit any files with the extension/suffix `.pyc`. It will help to set the preferences in your operating system so that extensions always appear.